TP10 : Algorithme de Kosaraju

Objectifs du TP :

- Implémenter l'algorithme de Kosaraju et l'analyser.
- Découvrir Printf et Scanf en Ocaml.

Cet énoncé s'accompagne d'un fichier source TP10_enonce.ml à compléter. On y trouvera en particulier un graphe G_{ex} sur lequel faire les premiers tests des fonctions à implémenter. Le fichier arxiv.txt servira à faire un test plus conséquent en fin de TP. Pour chaque question d'implémentation, on analysera la complexité de l'algorithme proposé.

Partie 1 *Algorithme de Kosaraju*

L'objectif de cette partie est de déterminer les composantes fortements connexes d'un graphe orienté et non pondéré. De tels graphes seront représentés par listes d'adjacence via le type suivant :

```
type graphe = int list array;;
```

1. Dessiner le graphe correspondant à G_{ex}. Lui appliquer l'algorithme de Kosaraju avec comme convention que dès qu'une alternative survient, on choisit de traiter en premier le sommet de plus petit numéro possible. Dessiner le graphe des composantes fortement connexes associé à G_{ex}.
2. Ecrire une fonction transposer de signature graphe -> graphe qui transpose le graphe en entrée.
3. Ecrire une fonction ordre_fin_traitement de signature graphe -> int list renvoyant la liste des sommets du graphe en entrée par ordre de fin de traitement décroissante dans un parcours en profondeur complet.
4. Ecrire une fonction numeroter_accessibles de signature graphe -> int list -> int array. Elle prendra en entrée un graphe G et une liste ℓ contenant tous les sommets de G (sans le vérifier). Elle renverra un tableau contenant en case i le numéro de la racine de l'arbre de parcours auquel appartient i dans un parcours en profondeur effectué en traitant les sommets dans l'ordre indiqué par ℓ.

   Indication : On pourra commencer par calculer à la main ce que devrait renvoyer numeroter_accessibles sur le graphe G_{ex}^T en parcourant les sommets selon l'ordre proposé par ordre_fin_traitement g_ex.
5. Déduire de ce qui précède une fonction kosaraju de signature graphe -> int array renvoyant un tableau cfc qui numérote les composantes fortement connexes du graphe G en entrée c'est-à-dire vérifiant que cfc.(i) = cfc.(j) si et seulement si les sommets i et j sont dans la même composante fortement connexe dans le graphe en entrée. Vérifier que kosaraju g_ex renvoie un résultat cohérent avec la question 1.
6. En déduire une fonction caracteristiques_graphe_quotient de signature graphe -> int*int qui associe à un graphe le nombre de ses composantes fortement connexes et la taille de sa plus grande composante fortement connexe dans cet ordre.
7. Est-il important que le parcours utilisé soit un parcours en profondeur dans ordre_fin_traitement ? Et dans numeroter_accessibles ?

Partie 2 *Test sur graphe réel*

On cherche à présent à tester les fonctions précédentes sur un graphe réel lu dans un fichier texte. Un tel graphe sera formaté (sérialisé en fait) de la façon suivante :

- La première ligne du fichier contient deux entiers n et p séparés par un espace avec n le nombre de sommets du graphe et p son nombre d'arêtes. Les sommets sont numérotés de 0 à $n-1$.
- Les p lignes suivantes contiennent chacune deux entiers x et y séparés d'un espace et symbolisant le fait qu'il existe dans le graphe un arc allant de x vers y. Il n'y a pas de répétition d'arcs.

8. Créer un fichier exemple.txt représentant le graphe G_{ex} dans ce format.

Les fonctions printf du module Printf et scanf du module Scanf permettent respectivement d'écrire sur la sortie standard et lire sur l'entrée standard de manière assez similaire aux fonctions de mêmes noms en C.

Plus précisément, Printf.printf prend en arguments une chaîne de caractère contenant un nombre $n \geq 0$ de formats suivie d'un nombre d'arguments égal à n et dont les types sont compatibles avec les formats de la chaîne et affiche la chaîne sur la sortie standard. La fonction Scanf.scanf prend en entrée un motif contenant éventuellement des formats et une fonction f, lit l'entrée standard en convertissant si c'est possible les morceaux de la chaîne lue correspondants aux formats en valeurs et applique la fonction f à ces valeurs.

Par exemple, si on compile (avec ocamlc par exemple) un fichier test.ml contenant les lignes suivantes :

```
let premier_argument, somme = Scanf.scanf "%d %d\n" (fun x y -> x, x+y);;
Printf.printf "%d %d\n" somme premier_argument;;
```

puis qu'on lance l'exécutable test obtenu dans un terminal, le prompt attend que l'utilisateur entre deux entiers séparés d'un espace, stocke dans premier_argument le premier entier donné et dans somme la somme des deux arguments donnés par l'utilisateur puis affiche ces deux quantités grâce à la deuxième ligne. On peut bien sûr rediriger le contenu d'un fichier sur l'entrée d'un tel exécutable afin de ne pas avoir à recopier à la main son contenu. Par exemple, si arguments.txt contient une ligne formée de 5 7, la commande

cat arguments.txt | ./test

devrait afficher 12 5 sur la sortie standard.

9. Ecrire une fonction lire_graphe de signature unit -> graphe permettant de lire un graphe écrit selon les modalités précédentes sur l'entrée standard. La tester sur exemple.txt.
10. Le fichier arxiv.txt contient une tout petite partie du graphe des publications sur la plateforme ArXiv : les sommets en sont des articles et il y a un arc de x vers y si l'article x cite l'article y. Déterminer le nombre de composantes fortement connexes et la taille de la plus grande composante fortement connexe de ce graphe.